Tampereen ammattikorkeakoulu

Sovellusten ohjelmointi ja käytettävyys

Oppimispäiväkirja

Antti Venetjoki

SISÄLLYS

1 Viikkoharjoitukset 1

1.1 Android -ympäristön asennus ja Hello World

1.1.1 Android -ympäristön asennus

Asensin version Android Studiosta, joka sisälsi myös Android SDK ja tarvittavat työkalut. Asennusohjelma ohjasi automaattisesti SDK asennukseen. Huomasin, että prosessi asensi myös virtuaaliset Android-laitteet (AVD), joita käytetään sovellusten testaamiseen ilman fyysistä laitetta. Valitsin AVD Managerista laitteeksi Pixel 8 ja Android 12 -version, jossa on API-versio 31. Virtuaalikoneen käynnistämisessä huomasin, että laite voi olla hidas, mikä johtui osittain koneen resurssien rajoitteista.

1.1.2 Github-linkki

github.com

1.1.3 Todiste ohjelman ajosta

1.2 Jetpack Compose -tutustuminen

1.2.1 Github-linkki

github.com

1.2.2 Koodi ajettuna Android virtuaalikoneessa

1.3 Kotlin essentials – osa 1

github.com

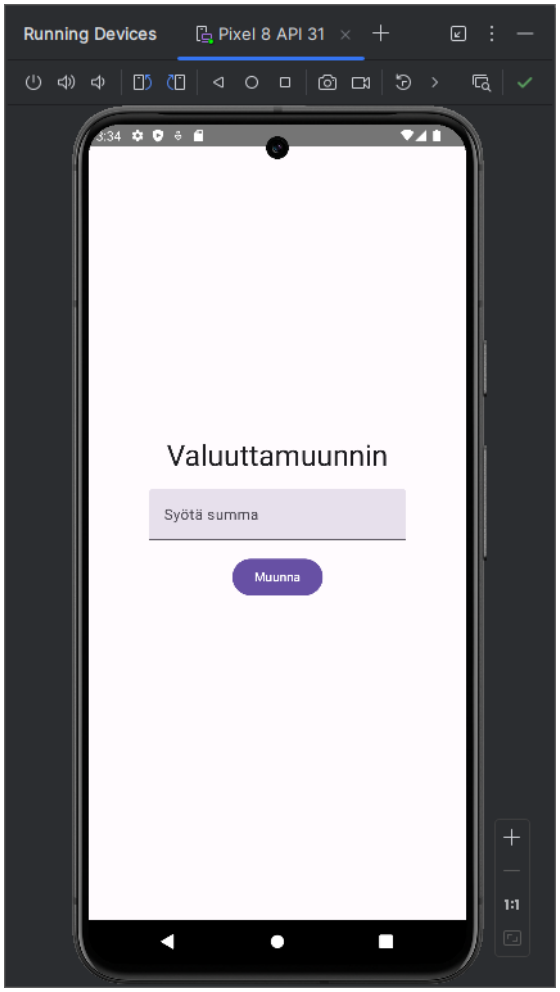
2 Viikkoharjoitukset 2

2.1 Valuuttamuuntimen käyttöliittymä

2.1.1 Github-linkki

github.com

2.1.2 Kuva käyttöliittymästä

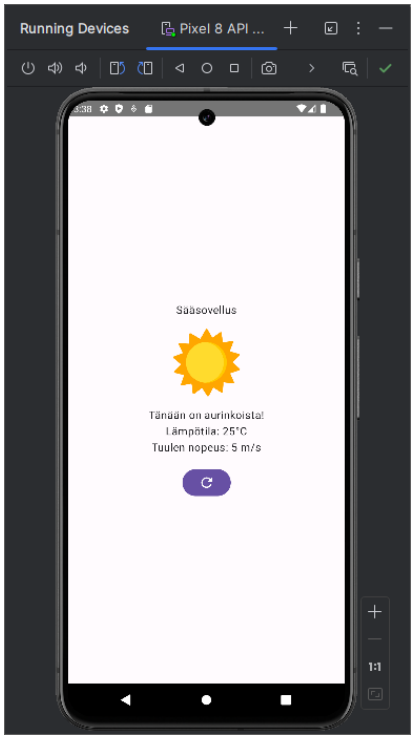


2.2 Sääsovelluksen käyttöliittymä

2.2.1 Github-linkki

github.com

2.2.2 Kuva käyttöliittymästä

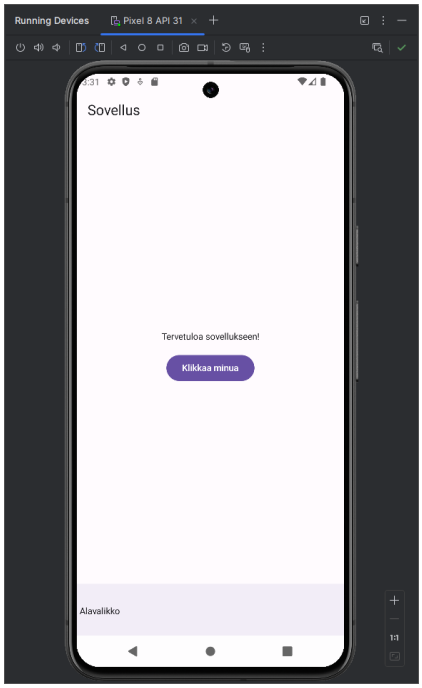


2.3 Scaffold

2.3.1 Github-linkki

github.com

2.3.2 Kuva käyttöliittymästä



2.4 Kotlin harjoituksia osa 2

github.com

3 Viikkoharjoitukset 3

3.1 Lokalisointi

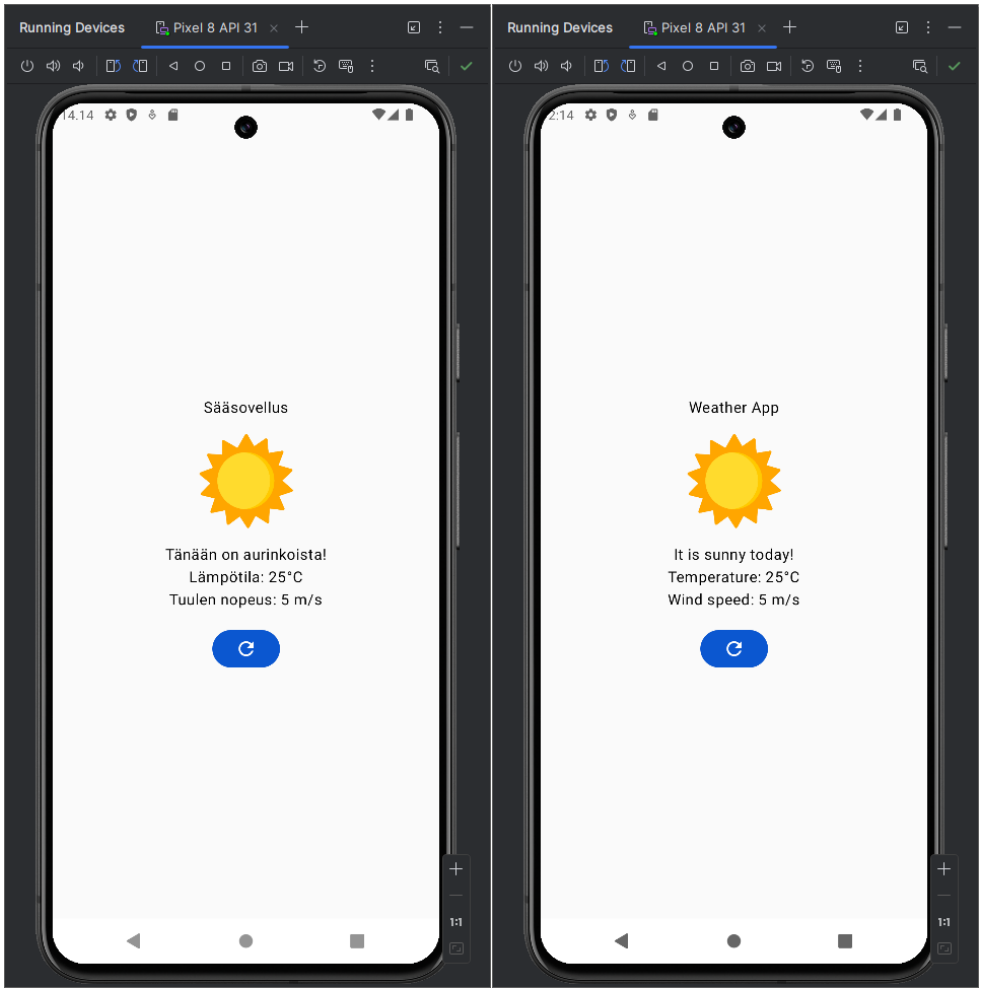
3.1.1 Pohdinta

Lokalisointi resurssitiedostoissa koodin sijaan on hyvä käytäntö monista syistä. Koodissa ei ole kovakoodattuja merkkijonoja, mikä tekee koodista siistimpää ja helpommin luettavaa. Resursseja voidaan muokata ilman, että kosketaan itse logiikkaan, mikä vähentää regressiovirheiden mahdollisuutta. Sovellus voi näyttää automaattisesti oikean käännöksen käyttäjän laitteen kieliasetusten mukaan, mikä parantaa käyttäjäkokemusta.

3.1.2 github-linkki

github.com

3.1.3 Kuvia ohjelman ajosta



3.2 Teemat

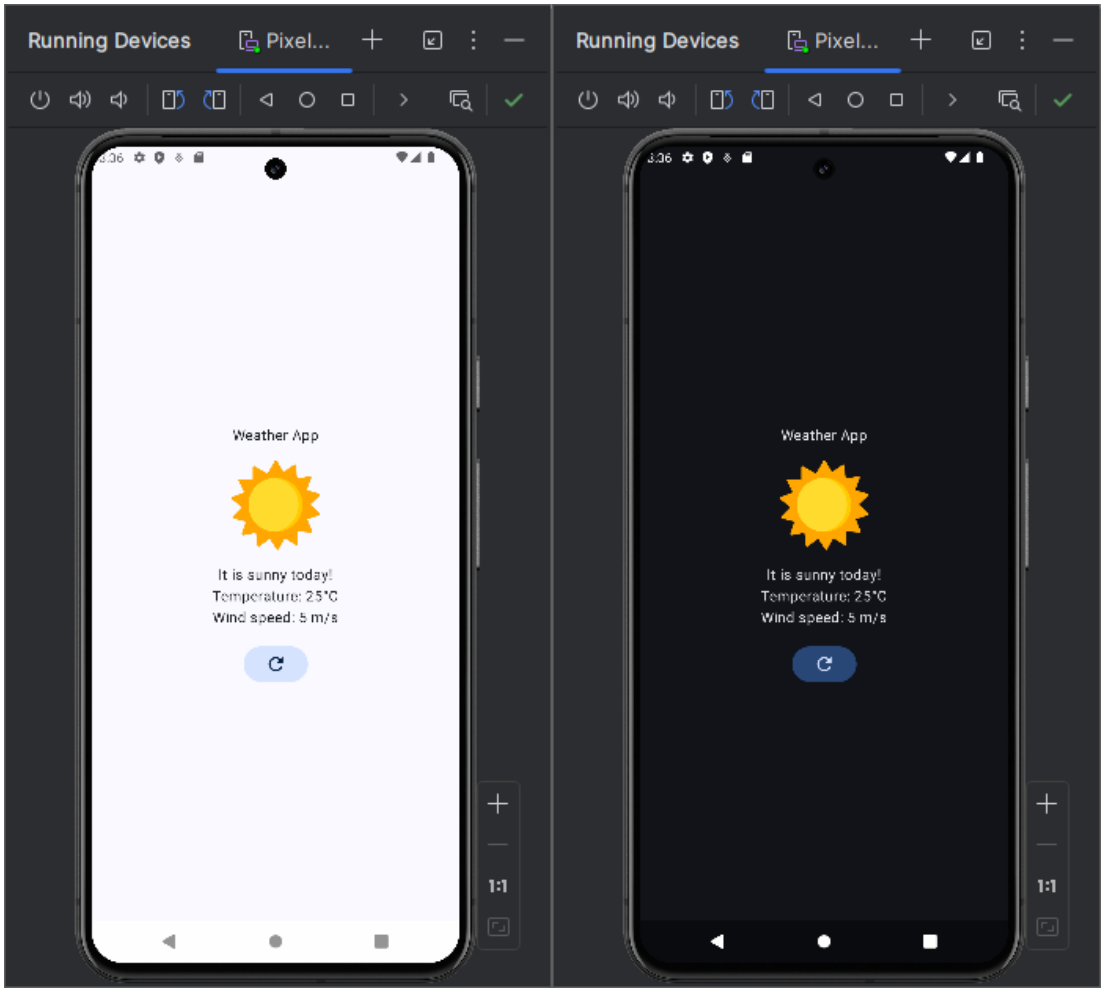
3.2.1 Pohdinta

Muutokset ulkoasuun voidaan tehdä helposti ja keskitetysti. Jos haluat muuttaa esimerkiksi kaikkien painikkeiden värin, voit tehdä sen teemasta käsin ilman, että sinun täytyy käydä läpi kaikkia käyttöliittymäkomponentteja erikseen. Teemat varmistavat, että koko sovelluksen ulkoasu on yhtenäinen. Väripaletti, fontit ja muut visuaaliset elementit pysyvät johdonmukaisina, mikä parantaa käyttäjäkokemusta. Teemat mahdollistavat tumman ja vaalean tilan tai jopa dynaamisten värien käytön Androidin materiaalidesignin mukaisesti. Sovellus voi mukautua käyttäjän järjestelmäasetuksiin tai antaa käyttäjälle mahdollisuuden valita eri teemoja itse.

3.2.2 Github-linkki

github.com

3.2.3 Kuvia ohjelman ajosta



3.3 Sovelluksen tila ja toiminnallisuus

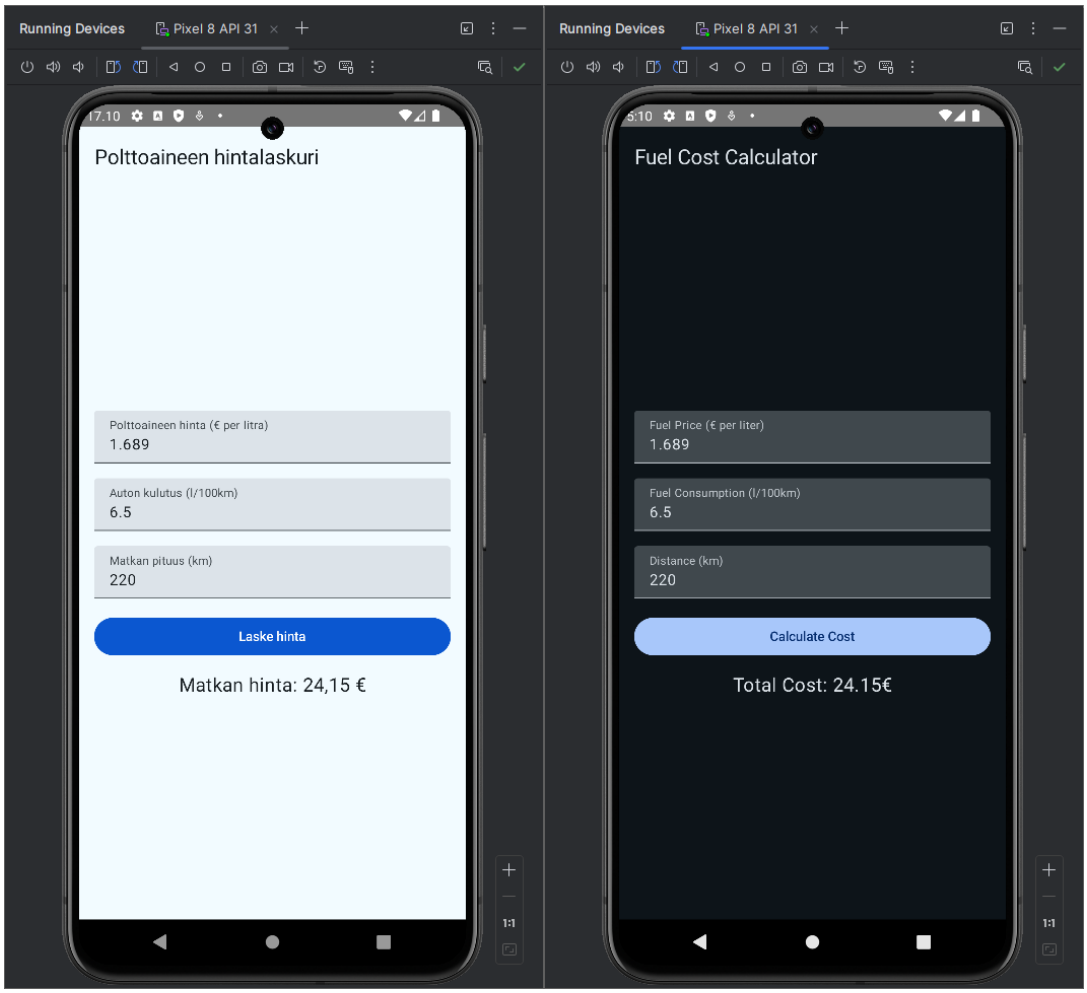
3.3.1 Pohdinta

Käyttöliittymän tilalla tarkoitetaan sovelluksen käyttöliittymän eri tiloja, joita voidaan käyttää käyttäjän interaktioiden seuraamiseen ja hallintaan. Näitä tiloja voi olla esimerkiksi käyttäjän syöttämät tiedot, sovelluksen näkymät tai eri elementtien tilat, kuten lomakekenttien sisällöt. Käyttöliittymän tilan hallinta on keskeistä, jotta sovelluksen eri osat voivat reagoida oikein käyttäjän tekemisiin.

3.3.2 Github-linkki

github.com

3.3.3 Kuvia ohjelman ajosta



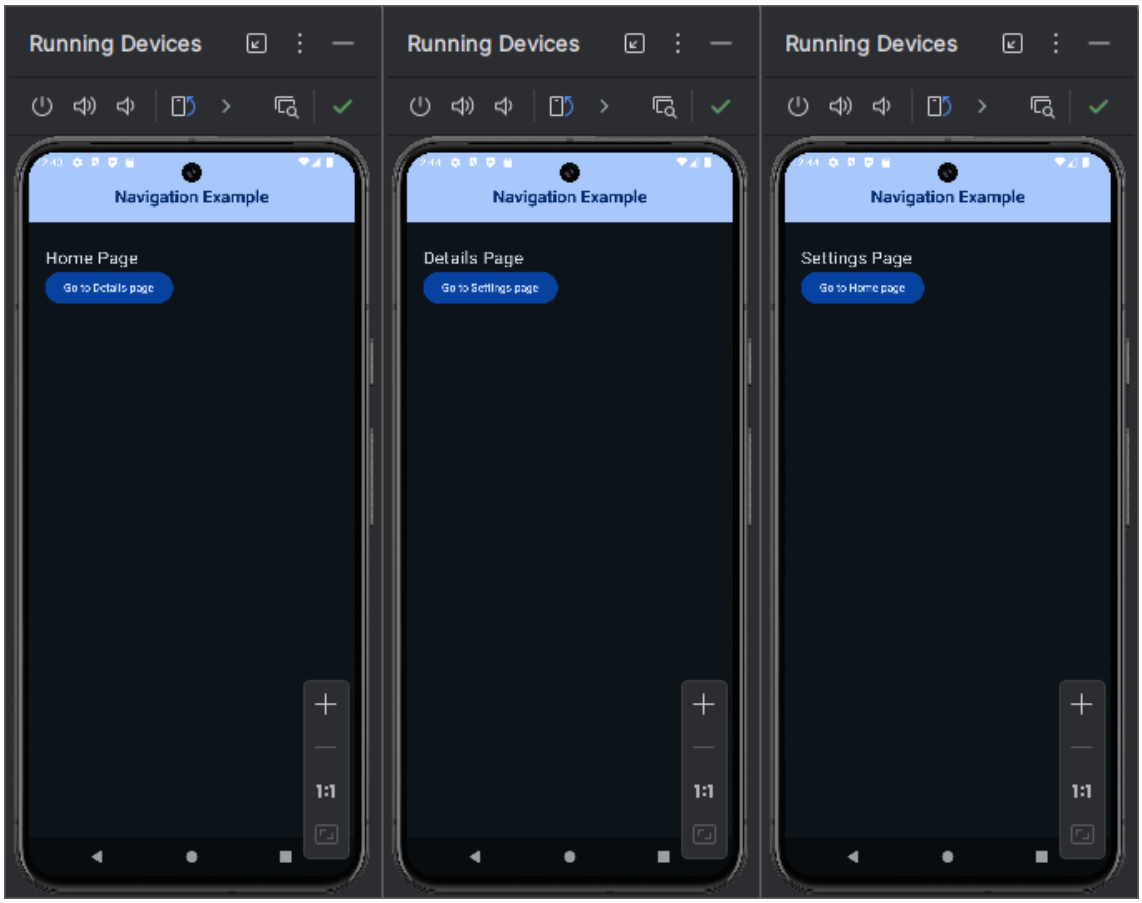
4 Viikkoharjoitukset 4

4.1 Navigointi

4.1.1 Github-linkki

github.com

4.1.2 Kuvia toiminnasta

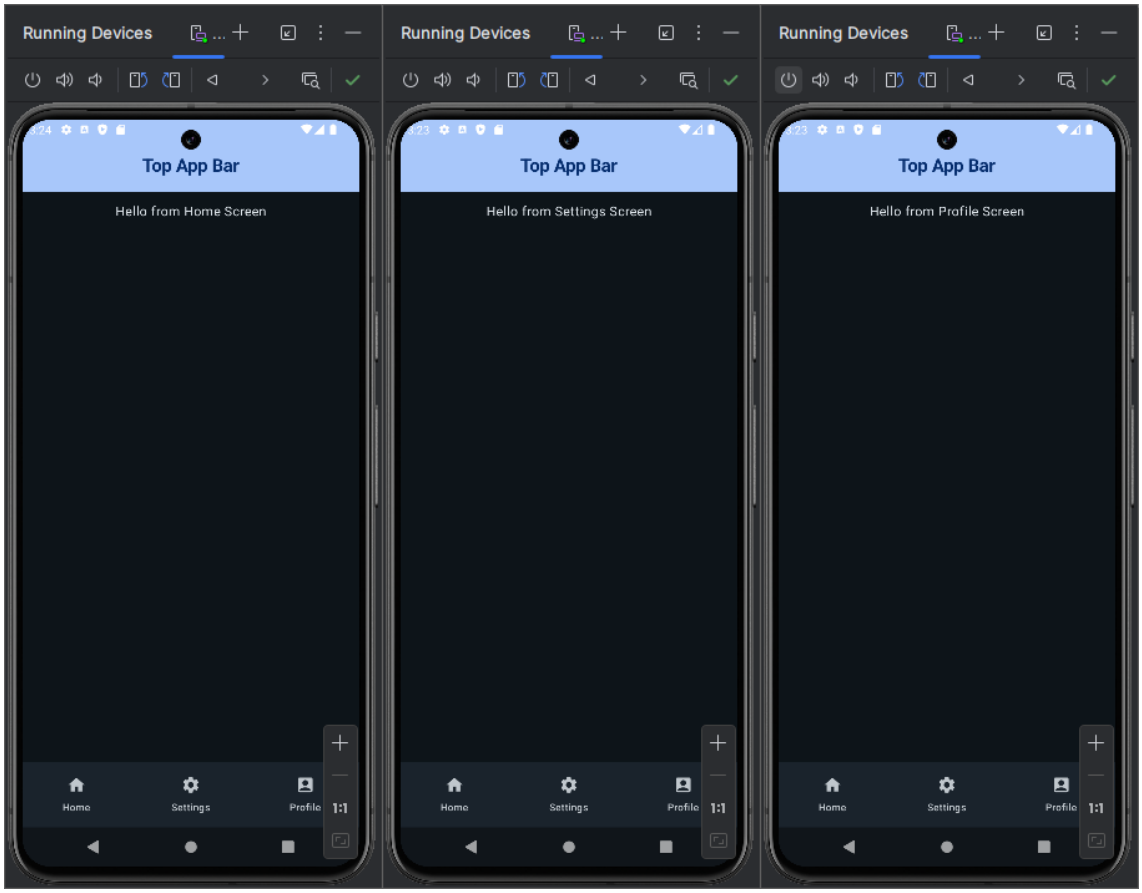


4.2 Bottom Tabs

4.2.1 Github-linkki

github.com

4.2.2 Kuvia toiminnasta



4.3 Intent

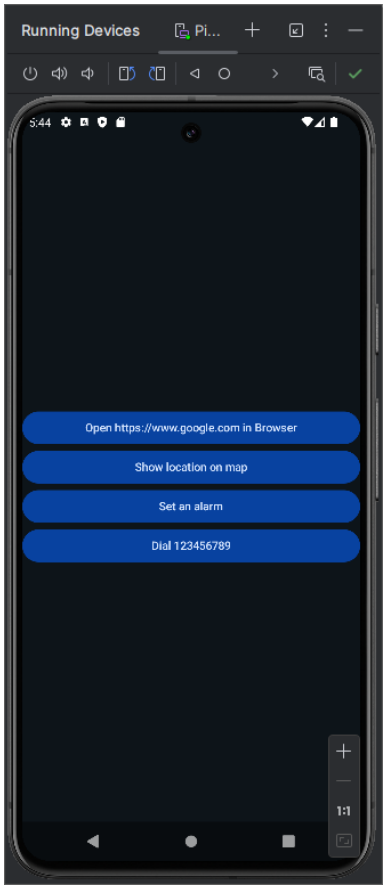
4.3.1 Pohdinta

Androidissa Common Intents tarkoittaa järjestelmän tarjoamia valmiita intenttejä, joita voidaan käyttää yleisiin tehtäviin sovellusten välillä. Intentti on olio, jota käytetään viestimään eri komponenttien (aktiviteettien, palveluiden jne.) välillä. Common Intents tarjoavat valmiita toimintoja, joilla sovellukset voivat käynnistää Androidin sisäänrakennettuja toimintoja tai siirtyä toisen sovelluksen tiettyyn toimintaan.

4.3.2 Github-linkki

github.com

4.3.3 Kuva ohjelmasta



5 Viikkoharjoitukset 5

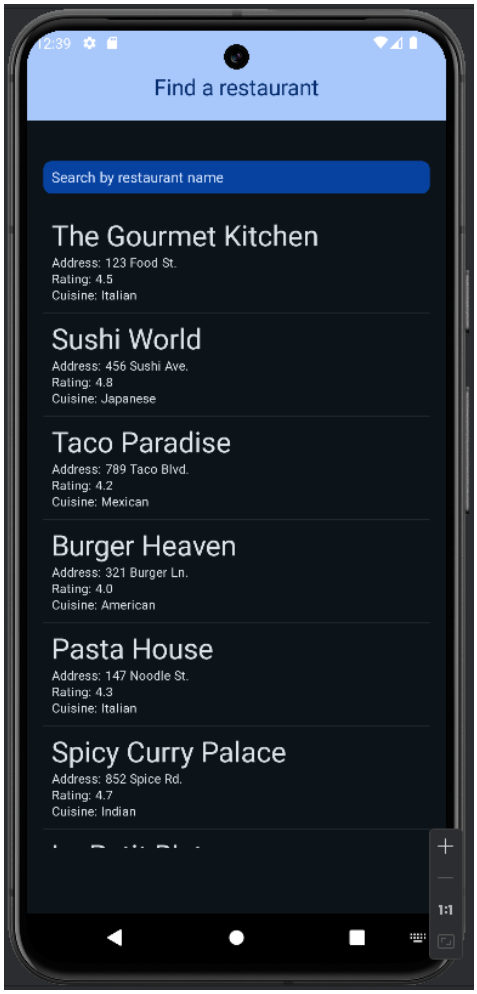
5.1 Dataluokat ja listojen toteuttaminen

5.1.1 Pohdinta

Kotlinin dataluokat (data classes) on suunniteltu tietorakenteiksi, ja ne generoivat automaattisesti tärkeät metodit kuten equals(), hashCode(), toString(), ja copy(). Javan dataluokissa nämä metodit täytyy kirjoittaa itse. Kotlinin dataluokat tukevat helppoa datan kopiointia ja toimivat sujuvasti data-binding-kirjastojen kanssa, kun taas Javassa vastaava käytettävyys vaatii lisäkoodia.

Column on yksinkertainen pystysuuntainen kontti, joka soveltuu pienille, kiinteille määrille elementtejä, sillä se renderöi kaikki elementit kerralla. LazyColumn on tarkoitettu suurille listanäkymille, ja se luo vain näkyvissä olevat elementit (lazy-loading), optimoiden suorituskyvyn ja muistinkäytön. Dynaamisissa listoissa LazyColumn on tehokkaampi vaihtoehto.

5.1.2 Kuva ohjelmasta



5.1.3 github-linkki

github.com

5.2 Detaljinäkymä

5.2.1 Kuva näkymästä

5.2.2 github-linkki

github.com

6 Viikkoharjoitukset 6

6.1 REST-toiminnallisuuden toteuttaminen Android-sovelluksissa

REST-toiminnallisuutta voidaan toteuttaa Android-sovelluksissa hyödyntämällä erilaisia lähestymistapoja ja kirjastoja verkkopyyntöjen tekemiseen. Kotlin-kielellä REST-pyyntöjen tekemiseen on useita vaihtoehtoja, kuten **HttpURLConnection**, **OkHttpClient**, **Volley** ja **Retrofit**. Jokaisella vaihtoehdolla on omat etunsa ja käyttötarkoituksensa.

6.1.1 Suorat HTTP-pyynnöt HttpURLConnection- ja OkHttpClient-luokilla

HttpURLConnection: Tämä on yksi Androidin perustyökaluista verkkopyyntöjen suorittamiseen. HttpURLConnection on yksinkertainen ja kevyt tapa toteuttaa perus-HTTP-pyynnöt, mutta sillä on rajoituksia, kuten monimutkaisuus virheenkäsittelyssä ja JSON-datan hallinnassa.

```
val url = URL("https://jsonplaceholder.typicode.com/posts")
val urlConnection = url.openConnection() as HttpURLConnection
try {
    urlConnection.requestMethod = "GET"
    val response = urlConnection.inputStream.bufferedReader().use { it.readText() }
    println(response)
} finally {
    urlConnection.disconnect()
}
```

Kuva 1 Esimerkki HttpURLConnection-pyynnöstä

OkHttpClient: Tämä on Googlen suosittelema vaihtoehto, joka tarjoaa useita edistyneitä ominaisuuksia, kuten asynkroniset pyynnöt ja tehokkaamman välimuistin hallinnan. OkHttpClient on luotettavampi ja joustavampi kuin HttpURLConnection, ja se on laajalti käytetty Android-kehityksessä.

```
val client = OkHttpClient()
val request = Request.Builder()
    .url("https://jsonplaceholder.typicode.com/posts")
    .build()

client.newCall(request).enqueue(object : Callback {
    override fun onFailure(call: Call, e: IOException) {
        e.printStackTrace()
    }

    override fun onResponse(call: Call, response: Response) {
        response.body?.string()?.let { println(it) }
    }
})
```

Kuva 2 Esimerkki OkHttpClient-pyynnöstä

Sovelluksen suorituskyky pääsäikeessä: Jos HTTP-pyyntöjä tehdään suoraan pääsäikeessä ilman erillisiä säikeitä tai asynkronista käsittelyä, se voi hidastaa käyttöliittymän toimintaa merkittävästi ja jopa johtaa sovelluksen "ei vastaa" -tilaan (ANR, Application Not Responding). Sekä HttpURLConnection että OkHttpClient mahdollistavat pyynnöt erillisessä säikeessä, mutta OkHttpClient tukee myös asynkronisia pyyntöjä, jotka ovat tehokkaampia ja vähemmän kuormittavia käyttöliittymän kannalta.

6.1.2 Volley-kirjasto REST-pyyntöjen toteutukseen

Milloin käyttää Volleya: Volley on Googlen kehittämä kirjasto, joka on hyvä vaihtoehto verkkopyyntöjen toteutukseen, erityisesti kun tarvitaan yksinkertainen ja kevyt ratkaisu ilman monimutkaisia konfiguraatioita. Se sopii pieniin ja keskisuuriin sovelluksiin, joissa on paljon verkkopyyntöjä ja mahdollisesti dynaamista tietoa.

Keskeiset ominaisuudet:

- **Välimuisti:** Volleyllä on sisäänrakennettu välimuisti, joka mahdollistaa aiempien pyyntöjen tallentamisen. Tämä parantaa suorituskykyä ja vähentää verkkokuormitusta.
- **JSON-tuki:** Volley tarjoaa sisäänrakennetun tuen JSON-objekteille, kuten JsonObjectRequest ja JsonArrayRequest, mikä helpottaa JSON-datan hakemista ja käsittelyä.
- **Asynkroninen pyyntöjono:** Volley hallinnoi automaattisesti pyyntöjonoa ja suorittaa verkkopyynnöt taustasäikeessä. Tämä varmistaa, ettei käyttöliittymän pääsäie kuormitu.

```
val url = "https://jsonplaceholder.typicode.com/posts"
val jsonObjectRequest = JsonObjectRequest(Request.Method.GET, url, null,
    { response -> println(response.toString()) },
    { error -> error.printStackTrace() }
)
Volley.newRequestQueue(context).add(jsonObjectRequest)
```

Kuva 3 Esimerkki Volleyn käytöstä JSON-pyynnön tekemisessä

6.1.3 Retrofit ja sen suosion syyt Android-kehityksessä

Miksi Retrofit on suosituin REST-kirjasto: Retrofit on yksi suosituimmista ja tehokkaimmista kirjastoista REST-toiminnallisuuden toteutukseen Android-sovelluksissa, erityisesti Kotlinin ja Jetpack Composen kanssa. Retrofitiä pidetään erittäin luotettavana ja kattavana, sillä se integroituu helposti Gson- tai Moshi-kirjastojen kanssa, jotka auttavat JSON-datan automaattisessa deserialisoinnissa suoraan Kotlin-luokkiin.

Retrofitiin liittyviä etuja:

- **Helppo JSON-käsittely:** Retrofit tukee automaattista konversiota JSON-datasta Kotlin-objekteihin, mikä vähentää koodin määrää ja virheiden mahdollisuutta.
- **Tuki erillisille HTTP-metodeille:** Retrofit tukee suoraan GET, POST, PUT, DELETE ja muita HTTP-metodeja annotaatiolla.
- **Asynkroninen käsittely ja coroutine-tuki:** Retrofit integroituu hyvin Kotlinin coroutinen kanssa, mahdollistaen tehokkaan asynkronisen käsittelyn ja pääsäikeen keventämisen.

```
// Retrofitin määrittely
interface ApiService {
    @GET("posts")
    suspend fun getPosts(): List<Post>
}

val retrofit = Retrofit.Builder()
    .baseUrl("https://jsonplaceholder.typicode.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

val apiService = retrofit.create(ApiService::class.java)

// Käyttö coroutinessa
CoroutineScope(Dispatchers.IO).launch {
    val posts = apiService.getPosts()
    println(posts)
}
```

Kuva 4 Esimerkki Retrofitin käytöstä

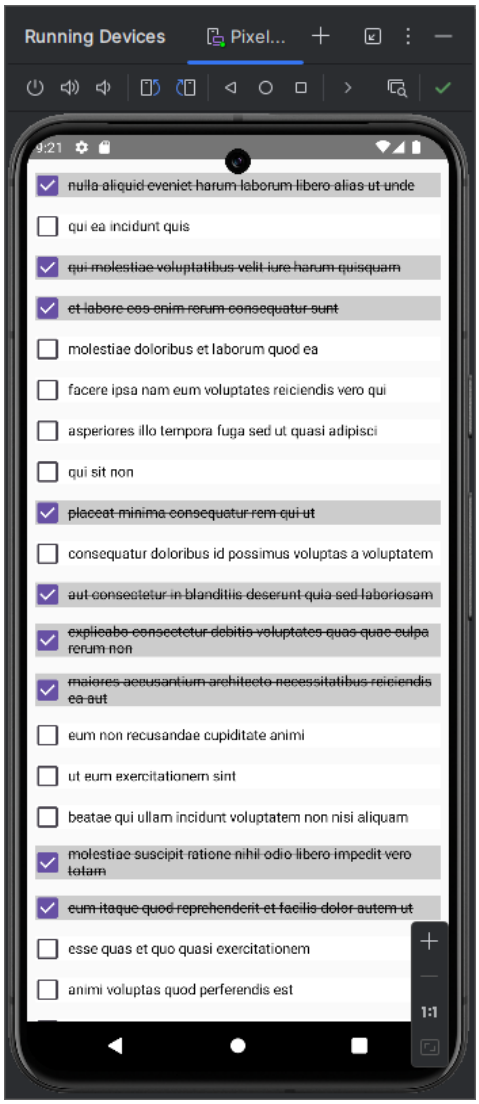
6.2 JSON-tiedon konvertointi Kotlin data-luokiksi

6.3 Tehtävälista-sovellus ja tietojen haku palvelimelta

6.3.1 Github-linkki

github.com

6.3.2 Kuva ohjelmasta



6.4 REST-pohjainen sääsovellus

6.4.1 Pohdinta

Sovellus on todettu toimivaksi ja sen jälkeen api-avain on poistettu tiedostoista. Tämän voisi korjata esimerkiksi .env tiedostolla, mutta sitä en tähän nyt alkanut toteuttamaan

6.4.2 Github-linkki

6.4.3 Kuva ohjelmasta

Käytetyt lähteet

https://kotlinlang.org/docs/

https://developer.android.com/guide

https://developer.android.com/studio/debug/

https://material-foundation.github.io/material-theme-builder/

https://carbon.now.sh

https://home.openweathermap.org